Custom Tags and Implicit Variables

1. List down the essential steps to writing a custom tag for use in your JavaServerPages?
    a. Writing a Java class called a tag handler
    b. Defining the tag within a tag library definition (TLD) file
    c. Providing details of where to find the TLD file in the deployment descriptor, web.xml
    d. Referencing the TLD file in your JSP page source and using the tags from it
2. Formerly—without EL and JSTL—you would always have to write your own ……. (Or acquire some from a third party) to avoid Java scripting.
    a. **custom tags**
3. List down the four "artifacts" involved with a custom tag?
    a. A Java class file, a TLD file, web.xml, and the JSP page itself.
4. Will the tag handler code have access to the JSP implicit variables?
    a. The tag handler classes don't. What they have instead is a PageContext object. We can use methods on this object to get hold of the implicit variable equivalents.
5. The pageContext implicit object of JSP is of type ……..
    a. javax.servlet.jsp.PageContext
6. Can we create objects of javax.servlet.jsp.PageContext and use it instead of pageContext implicit object?
    a. This is an abstract class. JSP container provides an instance of a subclass that has implemented all the abstract methods it contains.
7. How will you get the PageContext object in the tag handler code?
    a. The PageContext object is passed in the setPageContext() method. If your tag class inherits from TagSupport (or BodyTagSupport), setPageContext() saves the object to a protected instance variable so that you can access it directly in your code as pageContext.
8. How can you obtain different JSP implicit objects from the PageContext object?
    a. request
        i. getRequest()
    b. response
        ii. getResponse()

c. out
    iii. getOut() (inherited from PageContext's parent class JspWriter)
d. session
    iv. getSession()
e. config
    v. getServletConfig()
f. application
    vi. getServletContext()
g. page
    vii. getPage()
h. pageContext
    viii. This is the PageContext object passed to your tag handler
i. exception
    ix. getException()

9. An implicit variable that isn't necessarily always available is …….
    a. exception
10. When will the implicit variable 'exception' available within a tag handler class?
    a. The exception implicit variable is available only in a JSP error page. When another page specifies your error page and when that other page goes wrong, he exception produced by the other page is then available to your error page, as the implicit variable 'exception' in the JSP source. If that JSP source has a custom tag within it, that custom tag can invoke pageContext.getException() and expect an Exception object back. Under other circumstances, the call will return null.
11. How is **a "Simple" Custom Tag** Event Model different from a **"classic" tag** event model?
    a. Instead of three life cycle choices based on the interfaces Tag, IterationTag, and BodyTag for a "classic" tag event model, you have only one for a "Simple" Custom Tag Event Model, which is based on the javax.servlet.jsp.tagext.SimpleTag interface.

b. The doTag() method replaces all of doStartTag(), doAfterBody(), and doEndTag().All the processing that would have occurred in those methods moves to doTag(). Furthermore, doTag() returns nothing at all.
c. With simple tags, you don't use return codes (such as Tag.SKIP_BODY) to influence the life cycle. The life cycle as it directly affects the tag output is controlled instead by your code inside doTag().
d. Unlike classic model tags you get a new instance for every use of Simple tags. So you can safely initialize variables in your constructor or instance member declarations. There is no "pool" for simple tags.
e. In classic model, if the container decides to take a particular instance of a tag out of service, then release() is called, which you can use to clean up any expensive resources associated with the tag. There is no release() method for simple tags. So any cleanup must happen in doTag() (or a method called from doTag()).
f. In the simple tag file declaration within the TLD, the <body-content> is restricted to three allowed values: empty, tag dependent, and scriptless instead of four for custom model which also includes JSP.

12. What will happen if an attribute declared as mandatory in the TLD for a custom tag is missing from its use in the JSP? Need to find.
13. What will happen if an attribute declared as mandatory in the TLD for a custom tag is not having a corresponding setter in the tag handler? Need to find.
14. What are dynamic attributes?
    a. Attributes which can be added individually on a per tag and per JSP author basis are known as dynamic attributes.
15. How can a tag handler support dynamic attributes?
    a. A tag handler that supports dynamic attributes must implement the javax.servlet.jsp.tagext.**DynamicAttributes** interface implementing the **setDynamicAttribute**() method:
        i. **void setDynamicAttribute(String ns, String name, Object value)** – Container calls this to set the name and value of the attribute along with any preceding namespace qualifier. If qualifier is not given the attribute is assumed to be in the default namespace and ns will be null.

b. Additionally it must be configured in the TLD that this tag handler can accept dynamic attributes.
    i. <tag>
    ii. <name>MyTag1</name>
    iii. <tag-class>tags.Tag1</tag-class>
    iv. <body-content>empty</body-content>
    v. <dynamic-attributes>true</dynamic-attributes>
    vi. </tag>

16. What is variable synchronization?
    a. Variable synchronization is a mechanism through which container will update a variable in the JSP or the generated servlet to follow the value of a scoped attribute. Underneath it is always the scoped attribute which is modified by the tag. The synchronization causes the container to copy the value of the scoped attribute into the scripting variable, conveniently integrating JSP scripting elements with the tag.
    b. TLD is the primary location used to configure scripting variables and synchronization details. Tag files use the tag directive to specify the variables they expose.
    c. To declare synchronization in TLD we use the <variable> element in the TLD. Possible values are AT_BEGIN, NESTED, AT_END which is the scope of the scripting variable within the page.
        i. AT_BEGIN variables are created in the servlet code before the action is invoked and synchronized throughout the tags life cycle and remain in existence after the tag completes processing.
        ii. NESTED variables are created before the action is invoked and removed when the action terminates.
        iii. AT_END variables are created just after the tag finishes its invocation.
    d. Both classic and simple tags can expose variables for synchronization, but only classic tags can make use of them in their bodies as simple tags and JspFragment cannot contain JSP scripting.

17. During variable synchronization, will the container always declare the variables for synchronization? How can we change that behavior?
    a. During variable synchronization, the container will declare the variables for synchronization. However we can tell the container to re-initialize but not re-declare the variable by using a value of false for the <declare> element in the TLD.
    b. When using a value of false for the <declare> element for an attribute in the TLD, we should make sure that the variable already exist and if it doesn't exist, translation will fail.
18. If the container encounters an attribute which does not recognize for an action, what will it do first?
    a. It will first check the action's dynamic attribute setting in the TLD.
        i. If the action is configured to accept dynamic attributes, the JSP will translate and compile fine.
        ii. If the action cannot accept dynamic attributes, a validation error will occur.
19. Describe argument fragments?
    a. A JSP fragment is a portion of JSP code passed to a tag handler that can be invoked as many times as needed. A fragment can be considered as a template that is used by a tag handler to produce customized content. Unlike a simple attribute which is evaluated by the container, a fragment attribute is evaluated by a tag handler during tag invocation.
    b. Attributes may be configured in the TLD as attribute fragments so that they are converted into JSP fragments which can be invoked to process their contents, rather than any other regular object type. To declare an attribute as fragment attribute, we can use **<fragment>true</fragment>** in the TLD for that attribute.
    c. If you want to present an attribute as JspFragment to a tag, we will need to supply the attribute as a <jsp:attribute> child and not as a normal attribute.
    d. The JspTag handler for tags with fragment attributes requires a JavaBean setter method for each fragment attribute, just as for any other fixed attribute the tag

takes. The parameter to the setter method must be of type javax.servlet.jsp.tagext.JspFragment.

20. How does JSTL tags provide synchronization?
    a. JSTL tags do not provide synchronization forcing us to either do the synchronization ourselves or use EL.
21. Can a single JAR contain more than one tag library?
    a. A single JAR may contain more than one tag library each identified by a unique TLD.
22. Should a TLD accompany all tag handler classes and tag files within a JAR? What will happen if we have tag handler classes or tag files without a TLD?
    a. A TLD must accompany all tag handler classes and tag files within a JAR. Each TLD should be deployed into the META-INF directory or any subdirectory. Any tags or tag files not declared in the TLD will not be recognized or loaded by the container.
23. Is it valid for an explicit and implicit TLD to reference the same tag file?
    a. Yes
24. What is an implicit TLD?
    a. All tag files deployed directly into a WAR should be placed in the WEB-INF/tags or a subdirectory of it. Tag files deployed in the WEB-INF/tags or subdirectories of it are assigned an implicit TLD by the container. This implicit TLD is created by the container as and when required. An implicit TLD is created by the container for each subdirectory and each subdirectory therefore represents a different tag library.
25. Can we have a TLD created manually for a tag file?
    a. They need not have a TLD even though we can provide one.
26. How can we create implicit TLDs for our tag handler classes?
    a. Implicit TLDs will be created only for tag files, not tag handler classes. When deploying tag handler classes directly into the application, all tags must have an accompanying TLD inside WEB-INF or a subdirectory of it.
27. How are custom tags different from custom actions?

a. They are both the same. Custom tags are also known as tag extensions or custom actions.

28. What are tag handlers?
    a. The actual classes which implement and define the behavior of both classic tags and tag files are known as tag handlers. However in case of tag files, the tag handler class is created by the container during tag files translation and classic tag handlers should be programmed directly.
29. How is standard action's handler different from JSTL or custom action's handler?
    a. Unlike JSTL or custom actions, standard actions are optimized by the container and embedded directly into the translated servlet code, without using separate handler instances.
30. When packaging tag files into tag library JARs, where should the tag handler classes and tag files be present?
    a. When packaging tag files into tag library JARs, all tag handler classes are packaged into the JAR under a path which maps to their package names. All tag files must be placed in, or in any sub-directory of the META-INF/tags directory in the JAR. Tag files which are precompiled into tag handler classes are deployed like other tag handler classes.
31. The setXxx methods for each attribute xxx on the tag handler called before or after the doStartTag() or doTag() methods for custom tags and simple tags respectively?
    a. The container will call the setXxx methods for each attribute xxx on the tag handler always before the doStartTag() or doTag().

32. More Notes

a. When considering tag handlers, attributes can be considered to be the JavaBean's properties.
b. The tag handler can be almost considered as a JavaBean and should have atleast setters for these attributes.